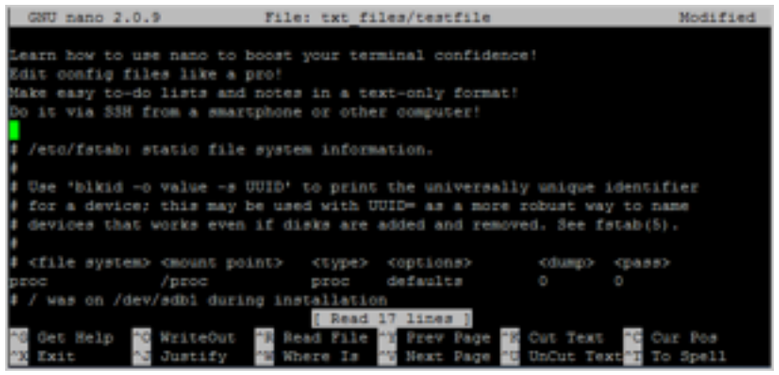
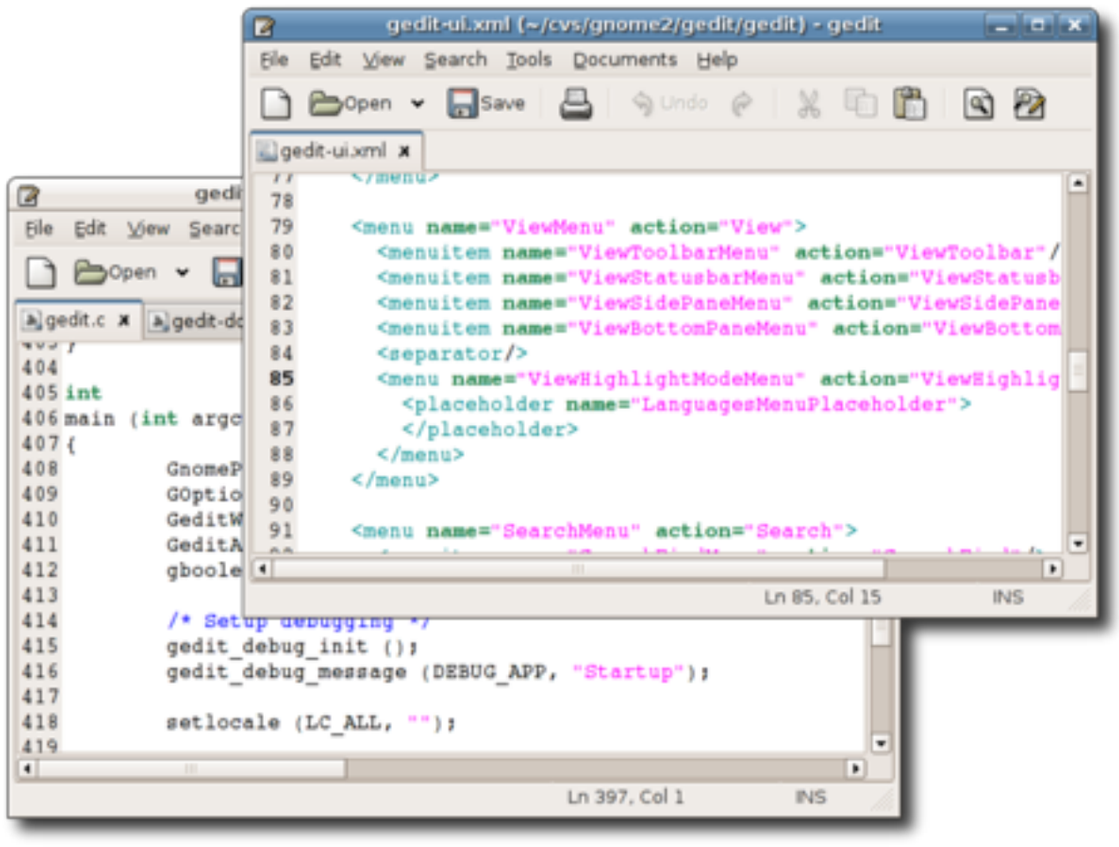
```
#!/bin/bash
```

Editores

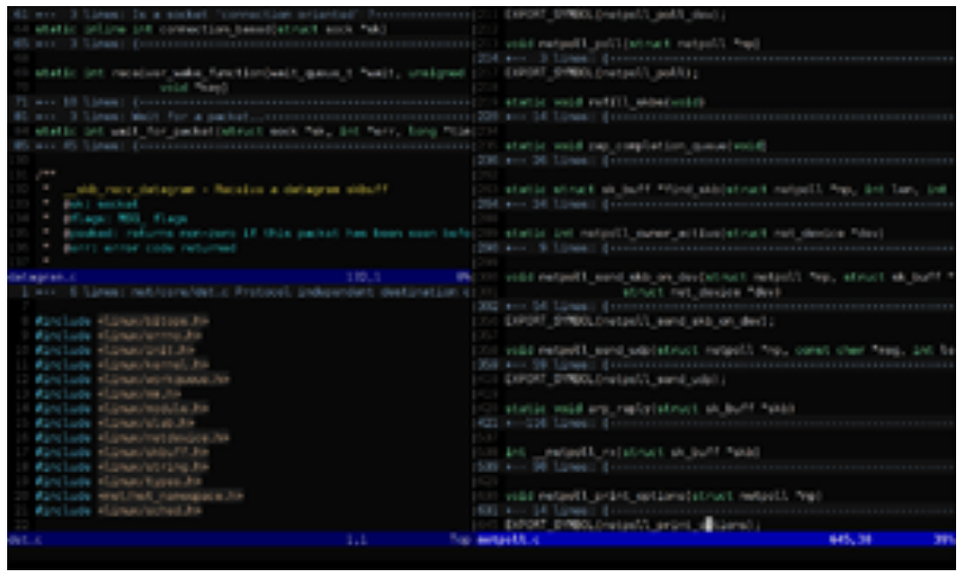
nano



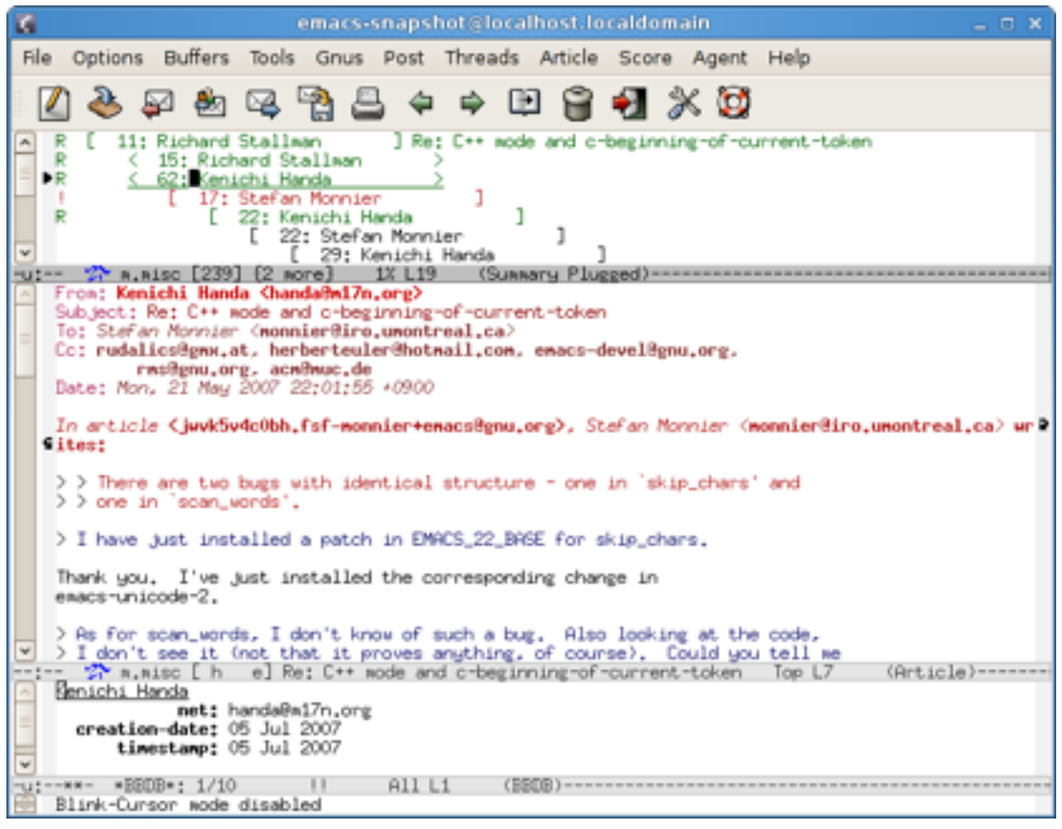
gedit



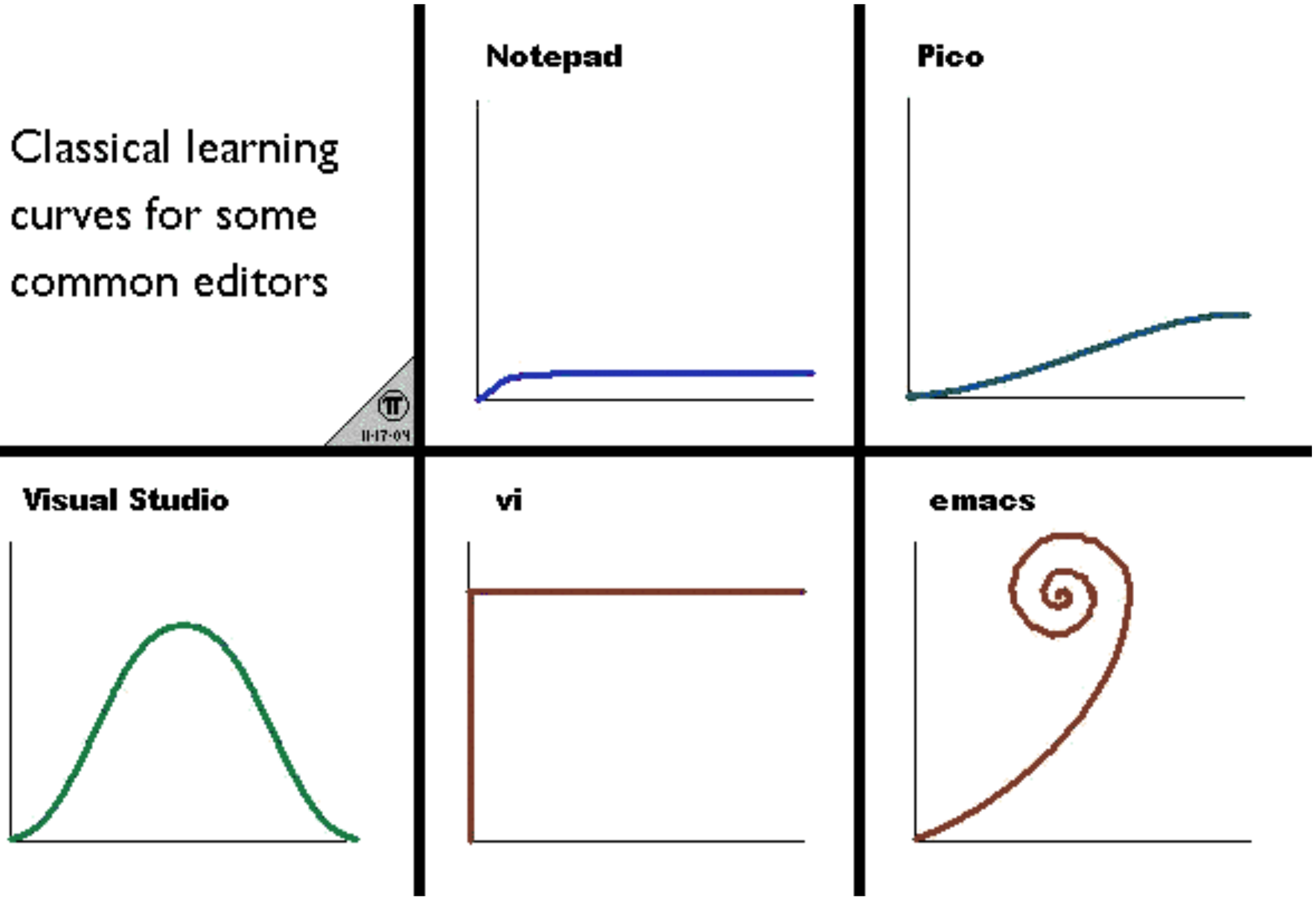
vim

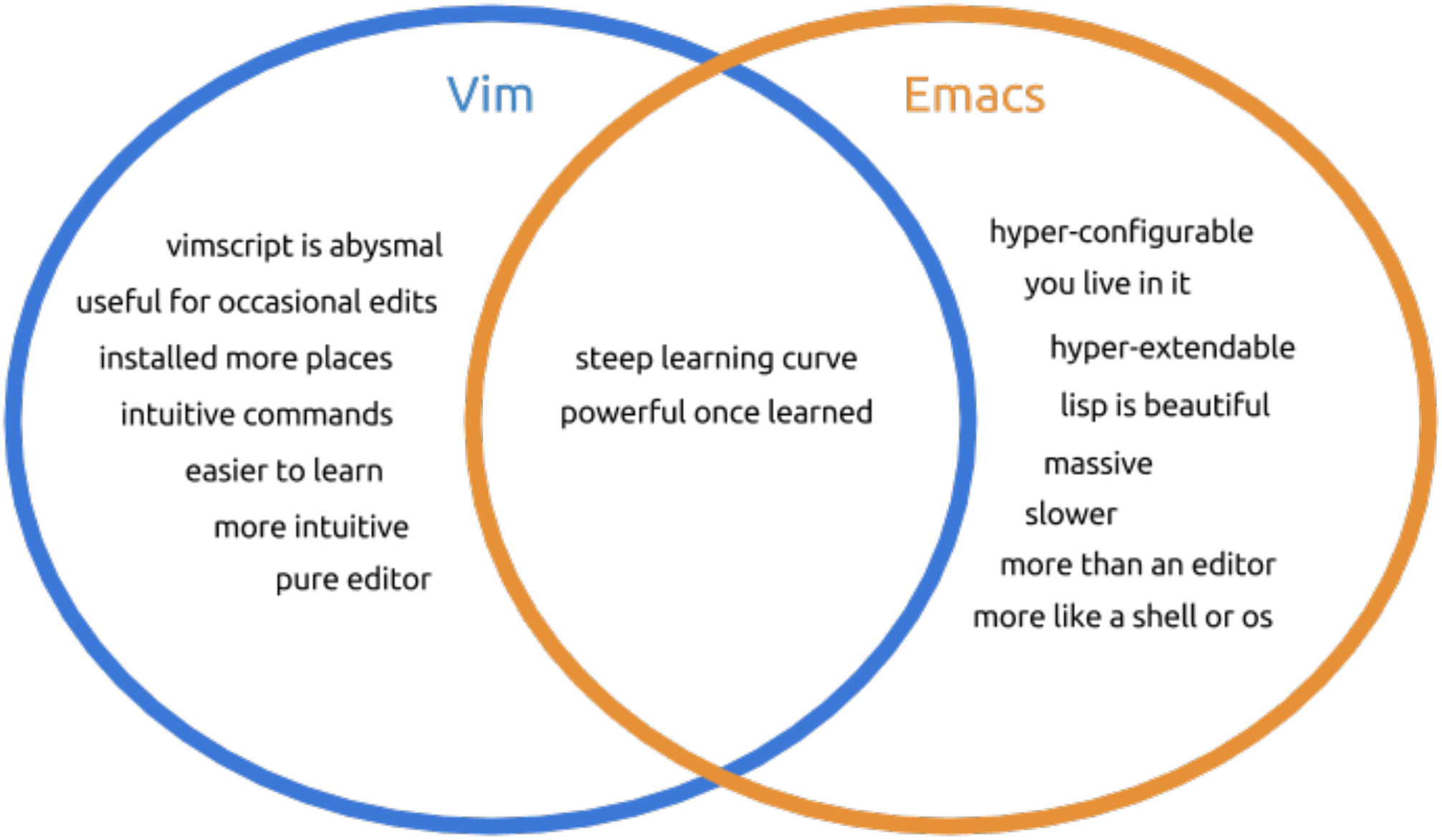


emacs



Classical learning curves for some common editors




Vim
Emacs
vimscript is abysmal
useful for occasional edits
installed more places
intuitive commands
easier to learn
more intuitive
pure editor
steep learning curve
powerful once learned
hyper-configurable
you live in it
hyper-extendable
lisp is beautiful
massive
slower
more than an editor
more like a shell or os

GNU Emacs Reference Card

(for version 20)

Starting Emacs

To enter GNU Emacs 20, just type its name: `emacs`

To read in a file to edit, see Files, below.

Leaving Emacs

suspend Emacs (or iconify it under X)	`C-z`
exit Emacs permanently	`C-x C-c`

Files

read a file into Emacs	`C-x C-f`
save a file back to disk	`C-x C-s`
save all files	`C-x s`
insert contents of another file into this buffer	`C-x i`
replace this file with the file you really want	`C-x C-v`
write buffer to a specified file	`C-x C-w`
version control checkin/checkout	`C-x C-q`

Getting Help

The help system is simple. Type `C-h` (or `F1`) and follow the directions. If you are a first-time user, type `C-h t` for a tutorial.

remove help window	`C-x 1`
scroll help window	`C-M-v`
apropos: show commands matching a string	`C-h a`
show the function a key runs	`C-h c`
describe a function	`C-h f`
get mode-specific information	`C-h m`

Error Recovery

abort partially typed or executing command	`C-g`
recover a file lost by a system crash	`M-x recover-file`
undo an unwanted change	`C-x u` or `C-_`
restore a buffer to its original contents	`M-x revert-buffer`
redraw garbaged screen	`C-l`

Incremental Search

search forward	`C-s`
search backward	`C-r`
regular expression search	`C-M-s`
reverse regular expression search	`C-M-r`
select previous search string	`M-p`
select next later search string	`M-n`
exit incremental search	`RET`
undo effect of last character	`DEL`
abort current search	`C-g`

Use `C-s` or `C-r` again to repeat the search in either direction. If Emacs is still searching, `C-g` cancels only the part not done.

Motion

entity to move over	backward	forward
character	`C-b`	`C-f`
word	`M-b`	`M-f`
line	`C-p`	`C-n`
go to line beginning (or end)	`C-a`	`C-e`
sentence	`M-a`	`M-e`
paragraph	`M-{`	`M-}`
page	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
function	`C-M-a`	`C-M-e`
go to buffer beginning (or end)	`M-<`	`M->`

scroll to next screen	`C-v`
scroll to previous screen	`M-v`
scroll left	`C-x <`
scroll right	`C-x >`
scroll current line to center of screen	`C-u C-l`

Killing and Deleting

entity to kill	backward	forward
character (delete, not kill)	`DEL`	`C-d`
word	`M-DEL`	`M-d`
line (to end of)	`M-0 C-k`	`C-k`
sentence	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill region	`C-w`
copy region to kill ring	`M-w`
kill through next occurrence of *char*	`M-z` *char*
yank back last thing killed	`C-y`
replace last yank with previous kill	`M-y`

Marking

set mark here	`C-@` or `C-SPC`
exchange point and mark	`C-x C-x`
set mark *arg* words away	`M-@`
mark paragraph	`M-h`
mark page	`C-x C-p`
mark sexp	`C-M-@`
mark function	`C-M-h`
mark entire buffer	`C-x h`

Query Replace

interactively replace a text string	`M-%`
using regular expressions	`M-x query-replace-regexp`

Valid responses in query-replace mode are

replace this one, go on to next	`SPC`
replace this one, don't move	`,`
skip to next without replacing	`DEL`
replace all remaining matches	`!`
back up to the previous match	`^`
exit query-replace	`RET`
enter recursive edit (`C-M-c` to exit)	`C-r`

Multiple Windows

When two commands are shown, the second is for "other frame."

delete all other windows	`C-x 1`	
split window, above and below	`C-x 2`	`C-x 5 2`
delete this window	`C-x 0`	`C-x 5 0`
split window, side by side	`C-x 3`	
scroll other window	`C-M-v`	
switch cursor to another window	`C-x o`	`C-x 5 o`
select buffer in other window	`C-x 4 b`	`C-x 5 b`
display buffer in other window	`C-x 4 C-o`	`C-x 5 C-o`
find file in other window	`C-x 4 f`	`C-x 5 f`
find file read-only in other window	`C-x 4 r`	`C-x 5 r`
run Dired in other window	`C-x 4 d`	`C-x 5 d`
find tag in other window	`C-x 4 .`	`C-x 5 .`
grow window taller	`C-x ^`	
shrink window narrower	`C-x {`	
grow window wider	`C-x }`	

Formatting

indent current line (mode-dependent)	`TAB`
indent region (mode-dependent)	`C-M-\`
indent sexp (mode-dependent)	`C-M-q`
indent region rigidly *arg* columns	`C-x TAB`
insert newline after point	`C-o`
move rest of line vertically down	`C-M-o`
delete blank lines around point	`C-x C-o`
join line with previous (with arg, next)	`M-^`
delete all white space around point	`M-\`
put exactly one space at point	`M-SPC`
fill paragraph	`M-q`
set fill column	`C-x f`
set prefix each line starts with	`C-x .`
set face	`M-g`

Case Change

uppercase word	`M-u`
lowercase word	`M-l`
capitalize word	`M-c`
uppercase region	`C-x C-u`
lowercase region	`C-x C-l`

The Minibuffer

The following keys are defined in the minibuffer.

complete as much as possible	`TAB`
complete up to one word	`SPC`
complete and execute	`RET`
show possible completions	`?`
fetch previous minibuffer input	`M-p`
fetch later minibuffer input or default	`M-n`
regexp search backward through history	`M-r`
regexp search forward through history	`M-s`
abort command	`C-g`

Type `C-x ESC ESC` to edit and repeat the last command that used the minibuffer. Type `F10` to activate the menu bar using the minibuffer.

HOW TO LEARN EMACS

A beginner's guide to Emacs 24 or later
Sacha Chua (@sachac) · sachachua.com/begin-emacs

Questions? I'd love to hear from you! 2013(v2)

STEP 0 (the geeky!)

If you're a developer or sysad...

Learn Vim → the other text editor

It's okay. Learn the basics so that you can easily work on other people's computers. If you know your way around Vim, people won't give you as much grief over Emacs.

Bonus mini-cheatsheet!
Here's what you need to know in Vi:
i insert mode ↔ <Esc> command mode
Long-time Vi user trying out Emacs?
:vimtutor
:w write/save file
:q quit
:q! really quit
evil mode Emacs

Emacs? You're never installing that on my box

You will probably meet editor zealots.

Be ambitextrous!

You can actually edit remote files in Emacs, but that's an intermediate topic. (Curious? See TRAMP)

server ssh/ftp emacs

Okay. Once you know the basics of Vim, you can get on with learning Emacs. ☺

Why Emacs?
- Amazingly customizable
- Endless room for growth

Most people start here!

STEP 1

Learn how to learn

There are some old books on Emacs, but the version differences can be rather confusing.

Start with the built-in tutorial instead.

Help → Emacs Tutorial ← repeat as many times as you need to

Other resources:
emacswiki.org
Lots of resources
planet.emacsen.org
Emacs-related blogs
IRC: irc.freenode.net → #emacs
Great for help and hanging out

Can't use the menu? Press Control-h t to start.

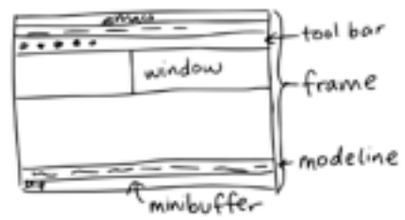
NOTE: The Emacs tutorial has lots of weird terms: "Meta key", "frame", "buffer". This is because Emacs started a long time ago. Don't worry, you'll get the hang of it with practice.

Some things that might help:

C-x C-s: This is how keyboard shortcuts are written.
① press Control & x at the same time
② then let go of x and press Control & s

Tip: Since you're using Control for both keys, you can hold Control down instead of letting go between x and s.

M-x lets you call commands by name, which is great if you can't remember the keyboard shortcut. Ex: M-x help-with-tutorial RET
(Meta usually Alt- or ⌘?) (This means press Enter/Return)



Windows show buffers, which could be
- files
- processes
- other info

Other good help commands:
C-h i manual
C-h k <keyboard shortcut> gives you help on it ☺
C-h f describes command/functions
C-h a searches for commands
C-h C-h shows help

STEP 2

Learn Emacs basics

C-x C-f open (find-file)
C-x C-s save
C-x C-c quit

NOTE: You don't need to quit Emacs after each file. Just leave it running and use C-x C-f to open the next.

How to select text
- Go to the start of your selection and press C-SPC (Control + Space)
- Go to the end of your selection and run your command

C-w cut (kill)
C-y paste (yank)
↳ M-y paste older things
C-/ undo

★ Learn how to use Keyboard macros. They're awesome.
C-x (start macro
C-x) end macro
C-x e execute macro
e ... again

STEP 3

Extend & customize

M-x load-theme RET
Try out color themes → use doku to see the list and list-packages to add more
M-x customize-group RET
set common options
M-x customize-face RET
change background, foreground, etc.

M-x list-packages RET
install lots of modules...
↳ Want a prettier Emacs? Check out color themes! (Lots of people like color-theme-solarized)

and then...

editing your ~/.emacs.d/init.el file!
↳ Huh? Where? Just use C-x C-f to create it!

initializes your Emacs, adds new functionality, and so on. } See emacswiki.org for lots of examples

Use M-x eval-buffer or restart Emacs to see the changes.

Broke your Emacs config? } emacs -q skips your customizations

STEP 4

Learn other handy tips

Buffer & window management
C-x b switch buffer → even better with M-x ido-mode
C-x 2 split
C-x 3 split
C-x o other window
C-x 0 get rid of current window
C-x 1 get rid of other windows

Navigation & Search
M-g M-g go to line (Hold Alt/⌘ and then press g twice)
C-s Interactive search
C-r Interactive search backward
M-> End of buffer
M-< Beginning of buffer
M-x occur RET Find lines

STEP 5

Explore!

Org-mode.org
organize your life in plain text

Narrowing/Widening focus

TRAMP remote access

Eshell / Term command-line in Emacs

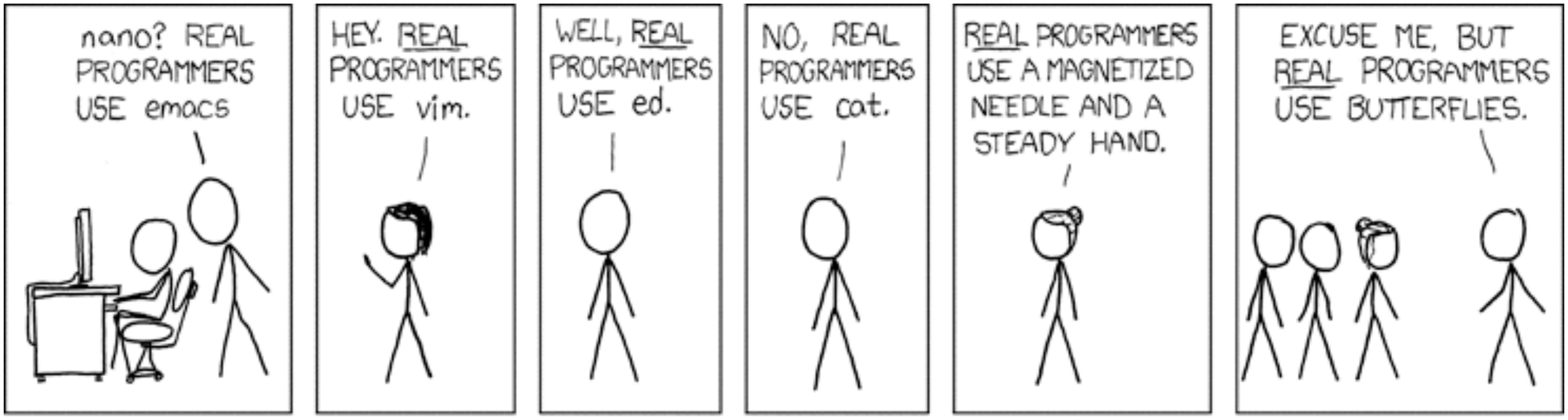
Calc powerful calculator and converter

Writing & debugging (+ editing) Emacs Lisp
(it sounds scary, but it's powerful ☺)

There's so much more!

Ask away and discover more by exploring!

Sacha Chua


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

> #!/bin/bash

BaSH Scripting

Un script es un archivo que contiene una serie de comandos a ser ejecutados por el shell. Éste puede interpretar dicho archivo así como cada entrada en la línea de comandos.

Podemos hacer estos scripts ejecutables agregando al principio del archivo una línea que le indica al sistema cuál es el programa que va a interpretar dicho script:

```
#!/bin/bash
```

Luego damos los permisos de ejecución necesarios

```
$chmod +x script.bash
```

Nuestro primer script:

```
#!/bin/bash
echo "¡Hola Mundo!"
```

```
$ chmod +x holamundo.bash
$ ./holamundo.bash
```

> #!/bin/bash

Variables

```
#!/bin/bash
MSG=“¡Hola Mundo!”
echo $MSG
```

Copia de seguridad simple

```
#!/bin/bash
BKFILE=~/mi-backup-$(date +%Y_%m_%d).tar.gz
tar -czf $BKFILE /home/pregrado
```

> #!/bin/bash

Variables

```
#!/bin/bash

var1=a var2=b
string1="dos palabras"
string2="$string1 y otras 2"
comando=$(seq 1 2 20)
operacion=$((7%5))

echo $var1 $var2
echo $string1
echo $string2
echo $comando
echo $operacion
```

> #!/bin/bash

Funciones

Son fragmentos de código con nombre que se pueden invocar posteriormente.

```
#!/bin/bash
function wait {
  echo "waiting..."
  sleep 2
}

echo "La terminal entra en modo de espera"
wait
echo "Bienvenido nuevamente"
```

> #!/bin/bash

Funciones

Pueden tener argumentos y retornar valores

```
#!/bin/bash
function suma() {
  resultado=$(($1+$2))
}

suma 3 2
echo "la suma entre 3 y 2 es" $resultado
```

La variable resultado definida en la función es una variable global, es decir, podemos acceder a ella desde cualquier parte del script.

> #!/bin/bash

Variables Globales y Locales

Podemos también definir variables locales, esto es, que solo existen dentro de la función:

```
#! /bin/bash
var=0
function func1 {
  local var
  var=1
  echo "la variable var en func1 es: $var"
}
function func2 {
  local var
  echo "la variable en func 2: $var"
}

echo "var es: $var"
func1
func2
echo "var sigue siendo $var"
```

> #!/bin/bash

Datos por stdin

Podemos pedirle datos al usuario por el stdin:

```
#!/usr/bin/env bash
echo "Ingrese su nombre"
read input
echo "Bienvenido $input"
```

```
> #!/bin/bash
```

Ejercicios

- Imprima la variable $0 en un script de BaSH
- Cree un script que imprima $1 y $2 y ejecute ./script.bash hola mundo
- Cree una función que duerma la cantidad de segundos dada como argumento.
- Modifique el script anterior para que el tiempo sea ingresado como argumento por línea de comandos.
- Cree un script que pida el usuario y revise si se encuentra en el sistema, dado el caso que reporte la ruta al home, cuantos carpetas tiene y cuantos archivos.
- El comando ps aux entrega varias columnas: usuario pid %cpu %mem vsz rss tty stat start time command. Cree un script que dado un usuario imprima: las tareas que está corriendo(solo el nombre, no al ruta completa); el PID; el uso de procesador y memoria.